# Towards Polyglot Adapters for the GraalVM

Fabio Niephaus
Hasso Plattner Institute,
University of Potsdam
Potsdam, Germany
fabio.niephaus@hpi.uni-potsdam.de

Tim Felgentreff
Oracle Labs
Potsdam, Germany
tim.felgentreff@oracle.com

Robert Hirschfeld
Hasso Plattner Institute,
University of Potsdam
Potsdam, Germany
hirschfeld@hpi.uni-potsdam.de

## ABSTRACT

Today, there are many different programming languages and even more software libraries and frameworks for various use cases. Polyglot runtime environments such as GraalVM allow developers to build and extend applications using multiple languages, which gives them a much broader choice in terms of frameworks and libraries available for reuse. Nonetheless, some usability problems remain, for example with regard to passing data from one language to another. GraalVM provides language interoperability through its polyglot API and allows objects and messages to be passed across languages. From a developer perspective, however, it is sometimes unclear *how* to pass non-primitive objects from one language into a library or framework written in another language. Code from that other language may expect these objects to respond to a different set of messages, which they may not understand at all.

In this paper, we present polyglot adapters, an early-stage concept that helps to pass objects across different languages. We explain how these adapters can improve the polyglot programming experience and demonstrate this with a prototype for the GraalVM.

## CCS CONCEPTS

• **Software and its engineering → Interoperability**; *Patterns*; *Object oriented languages.*

## KEYWORDS

GraalVM, polyglot programming, language interoperability


*Programming '19, April 1–4, 2019, Genova, Italy*

ACM ISBN 978-1-4503-6257-3/19/04…$15.00
https://doi.org/10.1145/3328433.3328458

**ACM Reference Format:**
Fabio Niephaus, Tim Felgentreff, and Robert Hirschfeld. 2019. Towards Polyglot Adapters for the GraalVM. In *Companion of the 3rd International Conference on Art, Science, and Engineering of Programming (Programming '19), April 1–4, 2019, Genova, Italy.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3328433.3328458

## 1 INTRODUCTION

Today, developers can use a wide range of programming languages for building software. The decision which language to use is often based on the skill set of a developer, but also on personal preference. In many cases, the latter is influenced by the libraries and frameworks available in a language. And once a language is chosen, it can be hard to re-use software artifacts from other languages.

Polyglot runtime environments such as GraalVM allow applications to be built using multiple languages. This gives developers a much broader choice in terms of libraries and frameworks they can use. Consequently, this allows them greater freedom in the choice of language, library, and framework for specific use cases.

Although projects like GraalVM support fast language interoperability, it is sometimes unclear *how* to use them for building polyglot applications, especially when data needs to be passed from one language to another. GraalVM supports to exchange objects between languages, which works well for values of primitive types [3]. As we will demonstrate, however, interoperability is not as seamless when exchanging non-primitive objects such as lists or data frames.

We present polyglot adapters, an early-stage concept supporting developers in passing objects across object-oriented (oo) languages. We explain how they can improve the polyglot programming experience and discuss advantages, disadvantages, and alternatives.

## 2 BACKGROUND

GraalVM [6] is a high-performance polyglot virtual machine developed by Oracle Labs. It provides language interoperability through its polyglot API which allows communication between objects through messages. For this to work, each language must be implemented in Truffle, GraalVM's language implementation framework. Since data from all languages must be represented internally as `TruffleObjects`, messages

can be sent even to data of languages that do not follow the OO paradigm. Moreover, the language implementers are responsible for implementing predefined interfaces required for language interoperability and must somehow expose the polyglot API within their language. At the time of writing, implementations for JavaScript, R, Ruby, LLVM, and Python are officially supported by GraalVM. All of them expose the polyglot API in a similar way: an *evaluate* API call allows evaluation of code of a certain language. Furthermore, it is possible to *export* data to a shared polyglot namespace which can then be *imported* again from any supported language.

## 3 APPROACH

We propose to apply the adapter pattern [1, pp. 139–150] in the context of polyglot programming to translate messages sent between languages. A polyglot adapter wraps around an object from a language and ensures that messages from other languages are understood correctly. This way, a non-primitive object does not need to be converted or reconstructed in another language before passing it into foreign code. Instead, it can be accessed directly from any other language. To make common use cases as easy as possible, we suggest that the adapters could provide a default mapping between languages for the most common object types. Additionally, it should be possible to configure and extend the behavior of polyglot adapters, so that developers can control how their objects behave when accessed from foreign code.

## 4 IMPLEMENTATION

We have implemented a prototype[1] of our polyglot adapters concept in Python 3 and demonstrate how they work in combination with GraalVM's polyglot shell[2].

Let's assume the Ruby function from Listing 1 has been exported to the shared polyglot namespace. Although this function can be imported in other languages via the polyglot API, calling it with objects from other languages may raise errors as shown in Listing 2. As the error message suggests, a Python object does not understand the `<=>` message which is sent to it as part of calling `ruby_func`.

This is where our polyglot adapters come into play. They intercept the message dispatch and translate the `<=>` message to an equivalent operation in Python. This message must be understood by all Ruby objects, but is not allowed by Python's syntax. Our adapters, however, are able to translate this message, so that Ruby's `sort` function can, for example, operate correctly on lists of Python lists as shown in Listing 3.

Moreover, the adapters can further be specialized depending on the wrapped object. For example, an adapter for a Python list object that is passed to JavaScript could automatically come with additional mappings that are typical for JavaScript arrays, so that they respond to messages like `push` or `reduceRight`.

Lastly, an adapter's behavior can be configured and extended in different ways. Our prototype supports to define new and override existing message mappings with other method names or Python lambda expressions. Listing 4 shows how a JavaScript function can be called with a Python list as well as an array from NumPy, a Python library for scientific computing. In case of the list, we extend the behavior of the adapter with an additional mapping for the message `map` and provide a Python lambda for computing the response. The NumPy array, on the other hand, already understands the message `map`. However, our adapters do not provide special default mappings for these objects, so it necessary to explicitly add a mapping for `map`. The value of this mapping is a string which instructs the adapter to look up the identifier in the original object. This way, it is possible to forward or redirect messages.

**Listing 1: A simple Ruby function exported to the polyglot namespace using GraalVM's polyglot shell.**

```
ruby> def ruby_func(*args) args.sort() end
:ruby_func
ruby> Polyglot.export_method(:ruby_func)
#<Method: Class(Object)#ruby_func <shell>:1>
```

**Listing 2: Calling the Ruby function from Listing 1 with Python objects results in an `AttributeError`.**

```
python> ruby_func = polyglot.import_value('ruby_func')
python> ruby_func([1, 3], [1, 2])
AttributeError: 'list' object has no attribute
  '<=>' (PException)
```

**Listing 3: Polyglot adapters ensure the Ruby function from Listing 1 can be called with any Python objects.**

```
python> from polyglot_adapters import as_ruby
python> ruby_func = polyglot.import_value('ruby_func')
python> ruby_func(as_ruby([1, 3]), as_ruby([1, 2]))
[[1, 2], [1, 3]]
```

**Listing 4: Behavior of polyglot adapters is extendable.**

```
javascript> Polyglot.export('squared',
    (array) => array.map((x) => x * x))
python> squared = polyglot.import_value('squared')
python> squared(as_js([1, 2, 3], extended_behavior={
    'map': lambda self, func: map(func, self)}))
<map object at 0x1883871b> # representing `[1, 4, 9]`
python> square_func(as_js(numpy.array([1, 2, 3])),
    extended_behavior={'map': 'map'})
<map object at 0x70d3cdbf> # representing `[1, 4, 9]`
```

[1] https://www.github.com/hpi-swa-lab/polyglot-adapters
[2] http://www.graalvm.org/docs/reference-manual/polyglot/

## 5 DISCUSSION

The first thing to note is that our concept requires developers to always explicitly request a new polyglot adapter for each object they want to pass to another language. Also, it is unclear how well the concept scales considering there are lots of different languages, language concepts, semantic differences, and object types. To further improve usability, these adapters could be defined once and then applied automatically to certain kinds of objects.

Moreover, our prototype is completely implemented on the language level. This allows developers to fully control the behavior of the adapters. A requirement for the implementation, however, is a mechanism to intercept message sends, which might not be supported by every language. In these cases, at least parts of the implementation could be moved to the level of the language implementation framework. Nonetheless, it needs to be possible to extend the default behavior somehow from within the language to support all kinds of objects.

An alternative solution would be to further extend the `TruffleObject` interface with more language concepts, but this, too, cannot anticipate all use cases. Thus, there would still be a need for customization on the part of the user.

As another alternative, objects could be converted between languages using serialization formats such as JSON. Otherwise, the developer is responsible for providing multiple implementations of the same data structures for all languages used as well as corresponding conversion mechanisms. Any serialization approach, however, comes with performance costs and requires coordination.

Another open question is related to the assumption that all sources are available. When sources of libraries or frameworks are unavailable, the concept of polyglot adapters might not be applicable, for example when it is unclear which messages need to be understood by each object.

Lastly, allocating polyglot adapters and intercepting message sends may impose performance overheads which need to be further investigated.

## 6 RELATED WORD

GraalVM provides proxy interfaces [4] as part of its SDK which allow Java applications to allocate objects of predefined kinds that, when passed to a guest language, are treated as objects of that language with special behavior. These interfaces are part of the Truffle framework and need to be supported by each language implementation.

Protocol Buffers [2] are used at Google to automate object serialization across languages. For this, a compiler can generate definitions and serialization code for various languages from a language-independent specification. This approaches the problem from the opposite perspective of sharing data inside of objects rather than the objects themselves.

Polyfills [5] are used in JavaScript as compatibility layers for technologies and language features across browsers and different versions of them. Therefore, they allow developers to use APIs regardless of whether they are supported by a browser or not. Similarly, polyglot adapters provide compatibility for objects of different languages.

## 7 CONCLUSION AND FUTURE WORK

We propose polyglot adapters, an early-stage concept which allows developers to pass non-primitive objects from one language to another. For this, polyglot adapters translate messages between languages and mimic language-specific behavior.

In the future, it would be interesting to see how these adapters could be used to facilitate interoperability between languages not following the OO paradigm. Furthermore, we want to investigate if these adapters could be automatically applied, for example from within the language implementation framework.

## ACKNOWLEDGMENTS

We gratefully acknowledge the financial support of Oracle Labs[3], HPI's Research School[4], and the Hasso Plattner Design Thinking Research Program[5].

## REFERENCES

[1] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1995. *Design Patterns: Elements of Reusable Object-oriented Software.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[2] Google. 2019. Protocol Buffers. https://github.com/protocolbuffers/protobuf

[3] Matthias Grimmer, Chris Seaton, Roland Schatz, Thomas Würthinger, and Hanspeter Mössenböck. 2015. High-performance Cross-language Interoperability in a Multi-language Runtime. In *Proceedings of the 11th Symposium on Dynamic Languages (DLS 2015).* ACM, New York, NY, USA, 78–90. https://doi.org/10.1145/2816707.2816714

[4] Oracle. 2019. GraalVM SDK Java API Reference. https://www.graalvm.org/sdk/javadoc/org/graalvm/polyglot/proxy/package-summary.html

[5] Remy Sharp. 2010. What is a Polyfill? https://remysharp.com/2010/10/08/what-is-a-polyfill/

[6] Thomas Würthinger, Christian Wimmer, Andreas Wöß, Lukas Stadler, Gilles Duboscq, Christian Humer, Gregor Richards, Doug Simon, and Mario Wolczko. 2013. One VM to Rule Them All. In *Proceedings of the 2013 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software (Onward! 2013).* ACM, New York, NY, USA, 187–204. https://doi.org/10.1145/2509578.2509581

[3] https://labs.oracle.com/

[4] https://hpi.de/en/research/research-school.html

[5] https://hpi.de/en/dtrp/